# The Oracle Database on HP Integrity servers

Executive summary ... 2

Resource management with Oracle and HP-UX ... 2
- Oracle's view of CPU resources ... 3
- Adding and deleting CPUs ... 3
- Parallel Query Slaves and Dynamic Processor resources ... 4
- Oracle Memory Management and dynamic memory movement ... 5
- Two different views of resource management ... 7
  - Oracle approach ... 7
  - HP-UX approach ... 8
- Oracle scheduler ... 8
- HP-UX scheduler ... 9
- Integration between HP-UX workload management and DBRM ... 10

Oracle and ccNUMA ... 12
- SGA and background processes ... 13
  - Background processes ... 14
- Shadow processes. ... 14
- Oracle shared servers ... 15
- Parallel query slaves ... 16
- ccNUMA and dynamic containers ... 16
- HP-UX 11.31 special considerations ... 17

Tuning a ccNUMA architecture by localizing memory accesses ... 17
- Tuning with a single listener ... 20
- Tuning with psets ... 20
- Oracle and ccNUMA, Summary ... 21

Conclusion ... 22

For more information ... 23

# Executive summary

This white paper describes Oracle® Database deployment on the HP Integrity server platform with a strong focus on HP-UX. Even though the focus of this paper is Oracle10*g*, much of it applies to Oracle11*g* as well, even though changes are to be expected, especially in the area of resource management.

The integration of HP-UX workload management products with the resource management features provided by the database is described. While the ability of HP-UX to dynamically reconfigure containers can be useful, care must be taken when these workload management products are used in an Oracle deployment.

The paper also outlines the relationship of the cache-coherent Non-Uniform Memory Access (ccNUMA) architecture provided by the HP Integrity server platform with database components such as the System Global Area (SGA), private server processes; Oracle shared servers and parallel query slaves. Since tuning opportunities with Oracle on ccNUMA tend to focus on maximizing local memory accesses, a methodology for configuring an Oracle instance to maximize the number of Cell Local Memory (CLM) accesses is provided. Note that this methodology has configurations that may make it more suitable for benchmarking than a typical production environment.

**Intended audience**: System Architects and Database Administrators responsible for Oracle deployments on HP-UX Integrity servers.

# Resource management with Oracle and HP-UX

This section focuses on how HP Virtual Server Environment (VSE) features work in an Oracle deployment – in particular, on the integration of HP workload management functionality with Oracle's database resource management products.

The following topics are covered

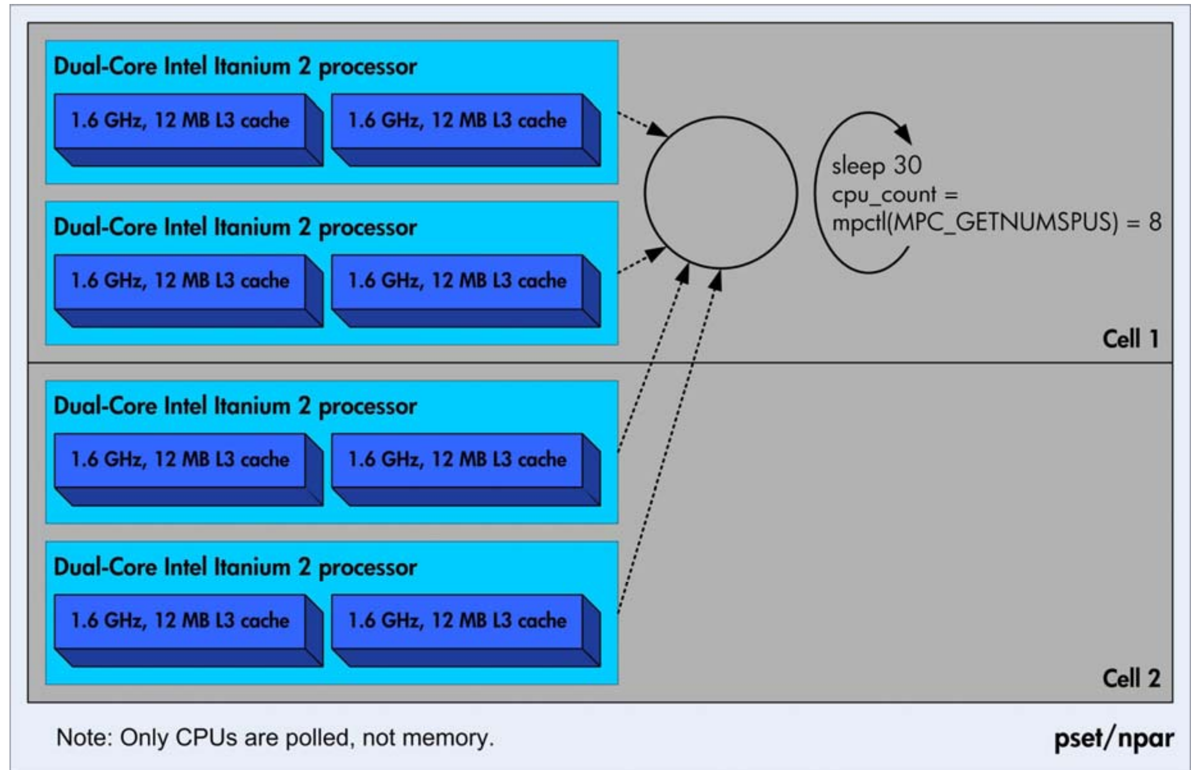
- Oracle's view of CPU resources
- Adding and deleting CPUs with HP-UX workload management products
- Partitioning memory into logical units with Process Resource Manager (PRM) Memory Resource Groups (MRGs)
- Two different views of resource management – at the database and OS levels
- An overview of Oracle and HP-UX schedulers
- A sample integration of Oracle Database Resource Manager (DBRM) and HP-UX Workload Manager (WLM)

VSE integrates the HP virtualization products on Integrity servers. This paper looks at VSE from the Oracle point of view: Resources (CPU, memory) being added to the container running the Oracle database and resources being deleted from the container. For more information about VSE, go to http://www.hp.com/go/vse. The container (or compartment) could be any of the supported compartments such as a Processor set (pset), a virtual partition (vPar), a hard partition (nPar) or a virtual machine (VM). Most of these containers types support dynamic adding and removing of resources.

## Oracle's view of CPU resources

Figure 1 outlines an Oracle database instance's view of the available computing resources within a system. In this context, we assume that the database instance runs within a dynamically reconfigurable container, such as a pset or a vPar.

**Figure 1.** Sample container




One thread constantly monitors the number of CPUs in the container in which Oracle is running. Internally, Oracle uses the HP-UX 'mpctl' call to determine the number of CPUs, which works very well for psets, vPars, nPars and VMs. In the case of FSS (Fair-Share Scheduler), however, using 'mpctl' does not work as well, since Oracle would be able to see all CPUs in the system, even though workload management policies might make fewer resources available. Oracle's internal algorithms are based on the number of processors, thus containers containing whole processors is a more natural choice for an Oracle deployment, especially when the DBRM is used.

## Adding and deleting CPUs

As indicated above, Oracle is prepared for the number of CPUs to change while the instance is running. On HP-UX, these changes would typically happen under the control of workload management products, Workload Manager (WLM) or Global Workload Manager (gWLM). Consult the web pages http://www.hp.com/go/wlm and http://www.hp.com/go/gwlm for more information on these workload management products. These products allow you to specify limits that can trigger the addition or deletion of CPU resources. For example, resources can be added if CPU consumption increases above the limit for the container in which Oracle is running; alternatively, if consumption falls below the specified limit, CPU resources may be removed and, if needed, given to other applications.

Note that CPU consumption is not the only metric that can be used to control the movement of CPU resources between containers; for example, the number of active Oracle users can be used to trigger the re-allocation of CPU resources.

Oracle's view of available CPU resources is published through the Oracle parameter 'cpu_count'. This parameter can also be set by the DBA, setting this parameter in the Oracle init.ora or spfile will override any dynamic changes in the container that Oracle runs in, and will make those changes invisible to the Oracle database. Oracle is dependent on the view of CPUs provided by the operating system. Hyper-threading, if enabled, is visible to Oracle as logical CPUs only – an 8 socket, dual-core partition or system with hyper-threading enabled will look like a 32 CPU server to Oracle.

The cpu_count parameter could be used to isolate Oracle instances from each other; on a 8 processor system, one Oracle instance could be given 6 processors by setting cpu_count = 6, and an other instance could be given the remaining 2 processors by setting cpu_count = 2. HP-UX, however, provides much better methods of isolating Oracle instances running on the same server through the use of psets or vPars, for example.

It is important, that Oracle has the correct view of available processor resources, as Oracle adjusts quite a few internal parameters based on cpu_count:

- parallel_max_servers: The degree of parallelism for parallel queries.
- Fast_start_parallel_rollback: This parameter controls the degree of parallelism for recovery of DML or DDL when you have a system crash.
- db_block_lru_latches: A too small value may result in contention on LRU latches.
- log_buffer: The size of the log buffer depends on the number of processors.

How many CPUs can be added to or deleted from an instance? Note that some of Oracle's internal data structures have dependencies based on the number of CPUs. Since these structures get allocated at database startup, adding too many CPUs to a running instance may provide enough resources for an excessive number of threads to run simultaneously, causing contention in the database. While there is no absolute rule, the following rule-of-thumb is offered:

> **The number of CPUs should increase by no more than three times the number of CPUs at database startup.**

It is still possible to add more processors, but the Oracle parameter cpu_count will not increase beyond this limit.

It is important to make sure, that an Oracle container does not loan out all but one processor, when Oracle is not running in that container. If this is the case, Oracle will only have one processor at startup, and this will put severe limits on how many processors that container can grow to.

In a ccNUMA aware deployment adding and deleting processors is a more complex task, the tools used need to be ccNUMA aware as well. Processors supporting a ccNUMA aware Oracle instance should run in a locality domain belonging to the Oracle instance. Processors from remote cells will not see the performance benefits of being close to a NUMA pool, which further restricts the choice of processors to add to a running Oracle instance. Currently the automated tools in HP-UX (gWLM, WLM) are not ccNUMA aware, and thus dynamic movement of CPUs using these tools may not always reach the desired results. See the "ccNUMA and dynamic containers" section for more information. The Oracle ccNUMA implementation assumes equally sized locality domains, so moving processors in and out may have undesired side-effects, since the likelihood of server processes migrating between locality domains increases.

## Parallel Query Slaves and Dynamic Processor resources

Above, we described the fact that deleting CPUs from an Oracle instance and adding CPUs to an instance will be observed by Oracle. If CPUs are added, Oracle processes will start running in the

new CPUs. However, if CPUs are added while a parallel query is running, the number of query slaves will not be increased for the running query. Only when the query has finished, and a new parallel query starts, will the new CPUs have effect. The default DOP (Degree of Parallelism) is 2 times the number of CPUs.

## Oracle Memory Management and dynamic memory movement

The discussion in this section applies to Oracle running without NUMA optimizations using interleaved memory for its SGA only. The ccNUMA aware implementation of Oracle splits up the SGA in equally sized NUMA pools across all locality domains in the container it is running in, and adding/deleting memory is not possible.

Even in the interleaved case adding and deleting memory is not a frequently used feature for the following reasons: 1) The feature requires swap space to be "overcommitted" to cover for future possible memory requirements and 2) Manual intervention is required to add and remove memory, and to change Oracle parameters to match the size of the memory available.

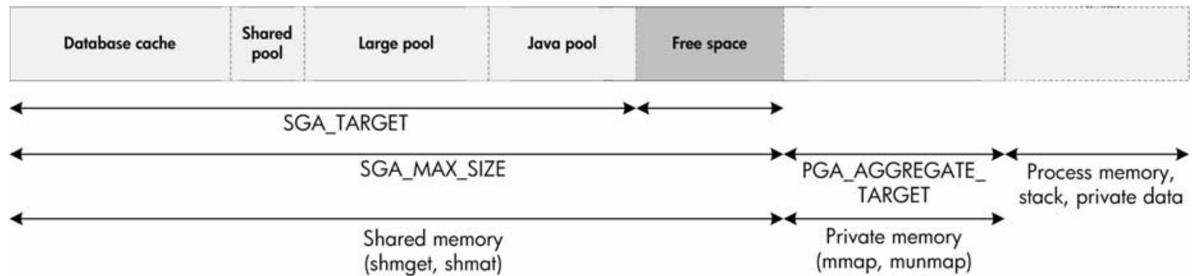
HP-UX shares the same memory management implementation as most other UNIX® platforms.

As of HP-UX 11i v1, the kernel implements Process Resource Manager (PRM) Memory Resource Groups (MRG) that allows you to partition memory into logical units. Each group is allocated a fixed amount of memory and has the option to borrow or lend memory, as needed. The latest version of PRM allows you to group shared memory as well, which makes it more suitable for Oracle deployments. The following web page provides more information about Oracle and MRGs: http://h20338.www2.hp.com/hpux11i/downloads/PRM.Oracle.SharedMemory.pdf

On HP-UX 11i v3, physical memory can be migrated between vPars as well, and Oracle can potentially take advantage of added memory in a vPar. For more about this feature, please consult the following website: http://docs.hp.com/en/9832/vParsMemMigration.pdf.

Oracle's memory management is outlined in Figure 2.

**Figure 2**. Oracle memory management




The following discussion applies to Oracle running in a non-ccNUMA system, or Oracle running with ccNUMA features disabled. A later section in this paper discusses the memory layout in a ccNUMA environment.

Oracle memory usage is split between the following:

- **Shared memory**
  The size of the shared memory – System Global Area (SGA) – segment used by Oracle is defined by SGA_MAX_SIZE. This parameter defines the total size of the shared memory segment(s) that Oracle will allocate and to which it will attach at instance startup. On HP-UX, Oracle uses the 'shmget(), shmat()' application processing interface (API) to manage this memory. On HP-UX, Oracle will allocate a SGA with the size of SGA_MAX_SIZE at database startup.

  SGA_TARGET controls how much memory is actually being used by Oracle and can be dynamically changed up to SGA_MAX_SIZE. Oracle automatically resizes the different pools within the SGA based on load and Automatic Workload Repository (AWR) advisories.

  On HP-UX, the SGA will often be locked in memory in many cases to improve performance: The administrator might have set the Oracle parameter 'LOCK_SGA' to true, and/or asynchronous IO on raw devices or volumes may be in use. The asynchronous driver /dev/async will also by default lock the SGA in memory. The impact of this is that there is really no "free memory", the whole SGA is in use immediately at startup. In cases, where the SGA is locked, it does not make sense to set SGA_TARGET < MAX_SGA_SIZE.

  If the SGA is not locked, the pages allocated for "Free space" in Figure 2 available for other use until SGA_TARGET is increased giving Oracle access to more shared memory; a shared memory segment sized at SGA_MAX_SIZE bytes is still allocated, but the area in the virtual space above SGA_TARGET will not be used. Swap space is still reserved for the whole segment, but since the area above SGA_TARGET is not used, thus it is "swapped" out, and does not occupy physical memory. The madvise(2) call is used by Oracle to tell HP-UX to free physical memory pages above SGA_TARGET.

- **Private memory**
  PGA_AGGREGATE_TARGET defines how much private memory – Private Global Area (PGA) – Oracle is allowed to use for all its active server processes and/or threads. On HP-UX, this memory is managed with 'mmap/munmap' calls, which means that memory actually gets released back to the OS when not needed.

  PGA_AGGREGATE_TARGET is defined at database startup. It can by changed with 'ALTER DATABASE set PGA_AGGREGATE_TARGET' while the database is running. This could be done for instance, if memory has been dynamically added to the partition Oracle runs in.

  Oracle private memory management can be turned off by setting the Oracle parameter '_use_realfree_heap' to 'FALSE'. This will cause Oracle to allocate memory with 'malloc' instead of 'mmap' and never free memory back to the OS. This is a higher performance method of memory management, but suitable for benchmarking only, and not recommended for normal use.

- **Other**
  Remaining memory usage consists of the normal overhead associated with running many processes (data segment, process stacks, and more).

Adding memory to a running Oracle instance would have the following impact:

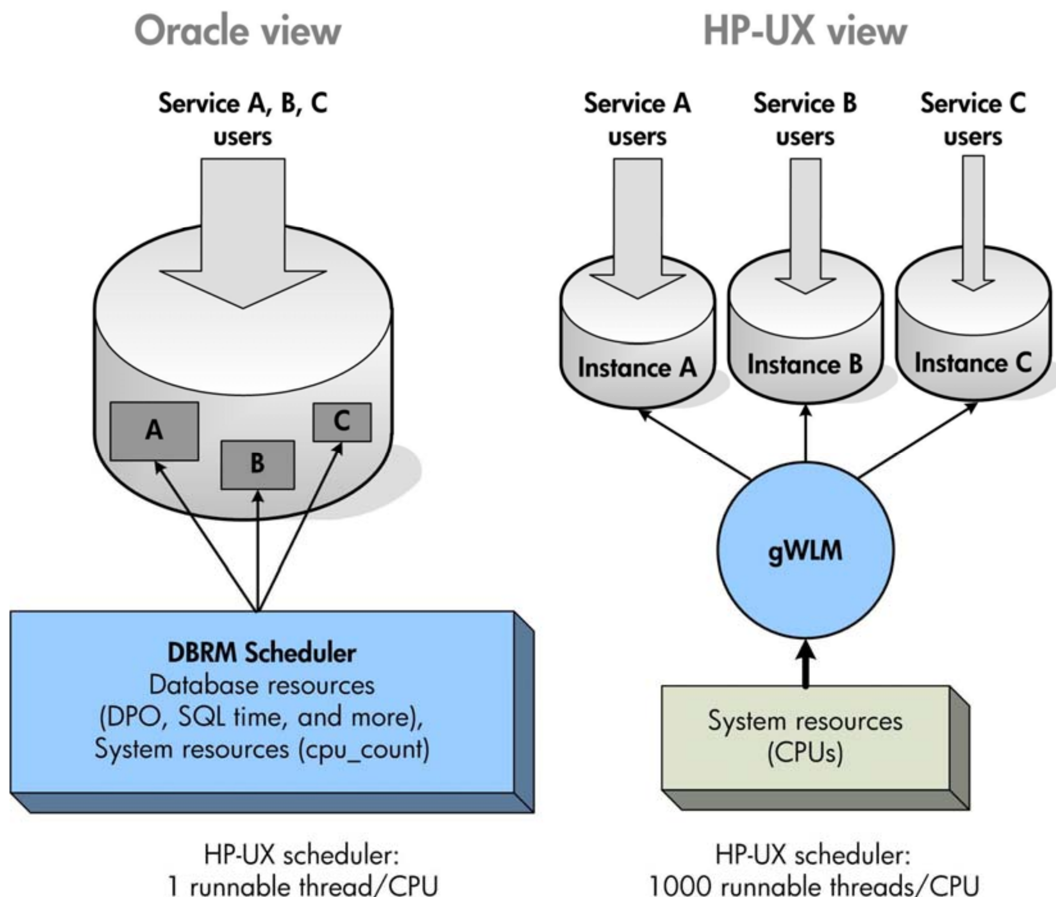
- It would automatically contribute to the area marked "process memory" in Figure 2, there would be space for more processes to run without paging.
- The DBA could increase SGA_TARGET allowing for more shared memory to be used by Oracle without paging out the SGA. Note that this will work only in the cases, where the SGA is unlocked.
- The DBA could increase PGA_AGGREGATE_TARGET allowing for more PGA use without increased paging

If there is a need to decrease memory, the SGA_TARGET and/or PGA_AGGREGATE_TARGET parameter values would have to be adjusted to meet the decrease in available memory as well.

## Two different views of resource management

Resource management in an Oracle database environment can be implemented at the database level, the OS level or both. Figure 3 provides two different views of resource management as provided by Oracle's Database Resource Manager (DBRM) and the HP-UX approach.

**Figure 3**. Different views of resource management




### Oracle approach

Oracle's resource management approach is based on a single database instance running many services. DBRM is aware of database resources and system resources (such as `cpu_count`) and is used to allocate resources to the different consumer groups.

A consumer group may have a one-to-one relationship with a service; in Figure 3, for example, Consumer Group A may consist entirely of users running Service A. However, consumer groups are purely DBRM entities that are not necessarily tied to specific services.

Benefits of the Oracle approach include:

- DBRM controls the allocation of database resources and can, for example, be used to restrict the degree of parallelism for table scan for lower-priority users.
- The time a query may consume can be restricted.

- Available CPU resources can be re-allocated between the different consumer groups, by giving 75% of CPU to Consumer Group A, for example.
- There is only a single database to administer.

**HP-UX approach**

HP-UX resource management tools can only allocate resources effectively if the services (A, B and C in Figure 3) are deployed on separate database instances. In this case, WLM/gWLM can move CPUs between the instances based on CPU utilization.

## Oracle scheduler

When activated, DBRM takes on the role of scheduling Oracle threads/processes. This is entirely user-space scheduling, with threads regularly asking DBRM for permission to run.

Figure 4 outlines this architecture.

**Figure 4**. The Oracle DBRM scheduler

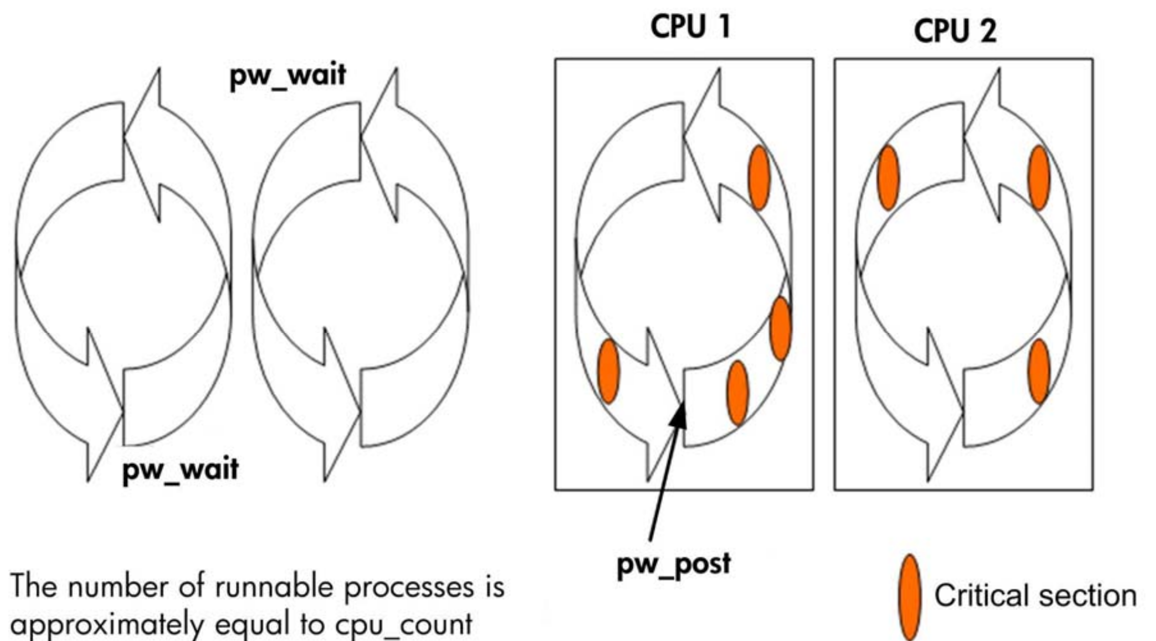



Figure 4 shows four Oracle processes running in a two-CPU container. Since DBRM tries to keep as many processes runnable as there are processors in the container, two processes are runnable; two are waiting for permission to run.

**Note:**
On HP-UX, a 'pw_wait/pw_post' mechanism is used to control the number of processes running.

At regular intervals the running processes ask for permission to continue; DBRM may choose to let the process run or place it in a wait state depending on CPU usage and the resource plans in effect.

The benefits of this scheduling approach include:

- An Oracle process/thread never gets interrupted by another process/thread while executing in a critical section holding a spinlock or latch, for example (marked in orange in Figure 4).
- Oracle makes scheduling decisions based on events that are only known to the database and not visible to any OS scheduler.
- Even on a very large system, the number of runnable processes is kept small, making the job of the OS scheduler easier.

In a production environment, the Oracle scheduler has to adapt to its processes yielding for reasons other than giving up time to other Oracle processes. Oracle server processes rarely keep the CPU 100% busy and may spend time blocked on I/O and/or inter-process communication (IPC); however, Oracle needs to keep the CPU busy under all conditions.

The DBRM uses the cpu_count parameter to decide how many threads to run in parallel.

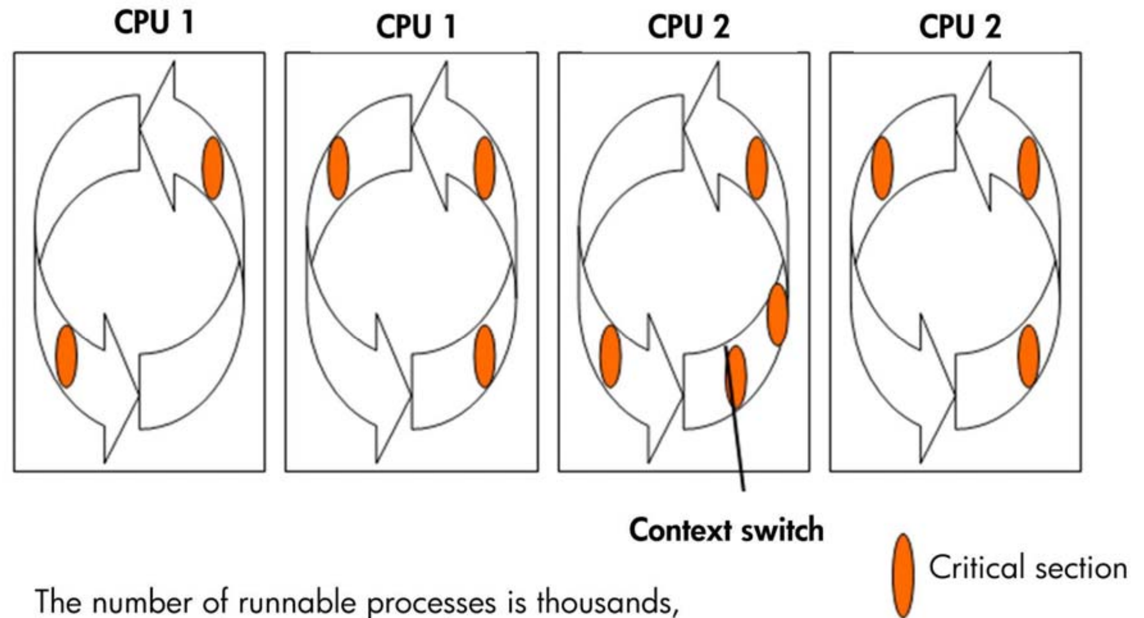
The Oracle scheduler probably works best in a container where Oracle has ownership of all CPUs, such as a pset, vPar, nPar or HP Integrity VM. In a FSS group, Oracle thinks it has access to all visible CPUs in the system and makes its scheduling decisions accordingly, and the DBRM scheduler may not work optimally, it may keep too many threads running. It is also a good practice to not let any other application occupy the same container as Oracle, to have the least possible impact on Oracle's scheduler. If processor sets are used, it is important to contain Oracle in one single processor set only, and not split up the database in several processor sets. If more than one processor sets are used, Oracle will have the incorrect view of available processors, and this may cause unnecessary contention in the database.

## HP-UX scheduler

If desired, scheduling tasks can be left to the HP-UX generic OS scheduler. However, since this scheduler is not aware of events happening within the database, it may not perform as effectively as the Oracle DBRM scheduler.

In practice, most HP-UX deployments still rely on the HP-UX scheduler, with good success. Moreover, while no performance comparison has been made between the two schedulers, benchmarks are still run using the HP-UX scheduler without DBRM.

**Figure 5**. The HP-UX scheduler




With the HP-UX scheduler, as shown in Figure 5, a large number of Oracle threads may be running in any CPU at any particular time. These processes keep running without needing to ask for permission. As a result, any process may be context-switched-out at any time, even when executing in a critical section. If the process were holding a latch when context switched-out, the release would take much longer, since that process would have to be re-scheduled before the latch could be released. This could potentially have a serious impact on overall system performance.

Different OSs deal with problems like context switches in different ways. On HP-UX, you should select a scheduler that can minimize risk by making sure that the priorities of processes do not degrade. Such degradation would extend the delay before the process could resume after being context switched out. Set the **`hpux_sched_noage`** parameter in Oracle's 'init.ora' file or 'spfile' as follows:

```
HPUX_SCHED_NOAGE = 178
```

178 is the highest **`sched_noage`** priority.

Note that the 'dba' group needs to have the RTPRIO privilege, to be able to set HPUX_SCHED_NOAGE.

The HP-UX scheduler is well integrated with the underlying hardware, and has all the knowledge about cores and hyper-threads to make intelligent scheduling decisions. This information is not visible to the Oracle user space scheduler; it only has a view of "logical" processors.

## Integration between HP-UX workload management and DBRM

Since HP-UX workload management products are unaware of Oracle's internal behavior, the integration of WLM/(g)WLM and DBRM to guarantee the best possible intelligent distribution of resources may present a challenge. For instance, if an Oracle instance is CPU-saturated by low-priority users, you may not want to allocate an instant capacity (iCAP) CPU to that instance and pay the extra price just to support low-priority users. On the other hand, if the CPU is being used by high-

priority users, you may indeed want to allocate iCAP resources to that instance so as to temporarily provide enough resources to support the users.

Currently, there is no simple integrated method for exposing Oracle database resource management information to the HP workload management products; however, it can be achieved through mechanisms such as the Oracle toolkit for WLM. You can execute a SQL script to find out the numbers of high- and low-priority users logged on, then set up a (g)WLM policy to react to the number of high-priority users and add CPU resources as needed. For example, you can set up a policy so that, if there are more than two high-priority users per CPU in the Oracle instance, add another CPU.

---

**Note:**
It is beyond the scope of this paper to provide detailed instructions for setting up this policy.

---

You can set up the following simple Oracle resource plan to give 75% of CPU resources to high-priority users ('orabm_priv') and the remaining 25% of users to low-priority users ('orabm_low'):

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_SIMPLE_PLAN(SIMPLE_PLAN => 'use_case_plan',
CONSUMER_GROUP1 => 'orabm_priv', GROUP1_CPU => 75,
CONSUMER_GROUP2 => 'orabm_low', GROUP2_CPU => 25);
END;
/
```

Now create a script to regularly survey and return the number of high-priority users and, through the 'wlmsend' utility, pass this metric to (g)WLM:

```
SELECT ACTIVE_SESSIONS
FROM V$RSRC_CONSUMER_GROUP
WHERE NAME = 'orabm_priv'
```

If (g)WLM policies have been set up to react appropriately, a CPU is added if there are more than or equal to two high-priority users per CPU in the Oracle instance, as shown in Figure 6.

**Figure 6**. DBRM/(g)WLM sample integration

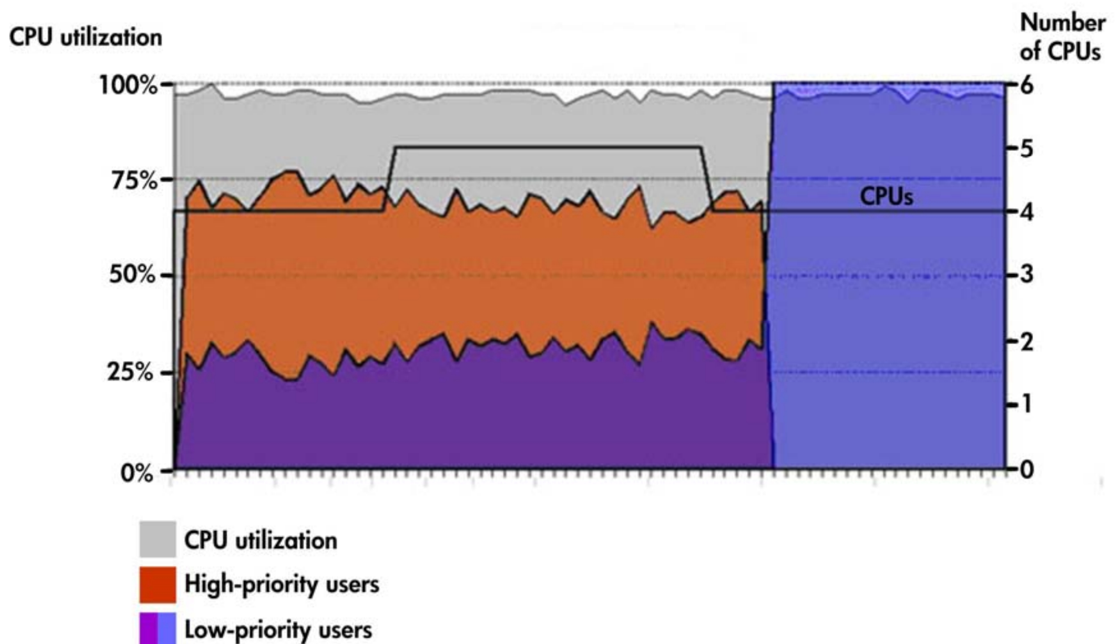



Figure 6 shows that, in this example, an Oracle instance (grey) is running at close to 100% CPU utilization with high-priority users (brown) consuming 75% and low-priority users (purple) consuming 25%.

With 7 high priority users, the number of high-priority users per CPU does not exceed two; four CPUs can accommodate the workload for these users. When an eighth high-priority user becomes active, the number of high-priority users per CPU now equals two; gWLM adds a CPU. With five CPUs, high-priority users continue to consume 75% of CPU resources per the resource plan.

When the eighth high-priority user again becomes inactive, the number of CPUs returns to four.

After the remaining high-priority users have completed their work, all available CPU resources are given to the low-priority users (blue). This is an important feature of the Oracle DBRM; low priority users are allowed to consume all resources not being used by high priority users.

A more elaborate DBRM resource plan, which is outside the scope of this white paper, could also restrict the number of low-priority users to a certain maximum number.

# Oracle and ccNUMA

Starting with the 10gR2 release, Oracle has begun building some ccNUMA-awareness into the database engine. This section outlines the impact of ccNUMA on database components such as the System Global Area (SGA), private server processes; Oracle shared servers and parallel query slaves. The ccNUMA optimizations are work in progress, and very little documentation exists. There is short mention in the Metalink note 399261.1 which outlines the architecture. The description in this paper is based on observations made on live systems, but may not cover all NUMA optimizations made by Oracle.

Current releases of Oracle will handle placement of shared memory (SGA) and Oracle background processes. For most of the other processes (shadows, shared servers, query slaves), Oracle relies on

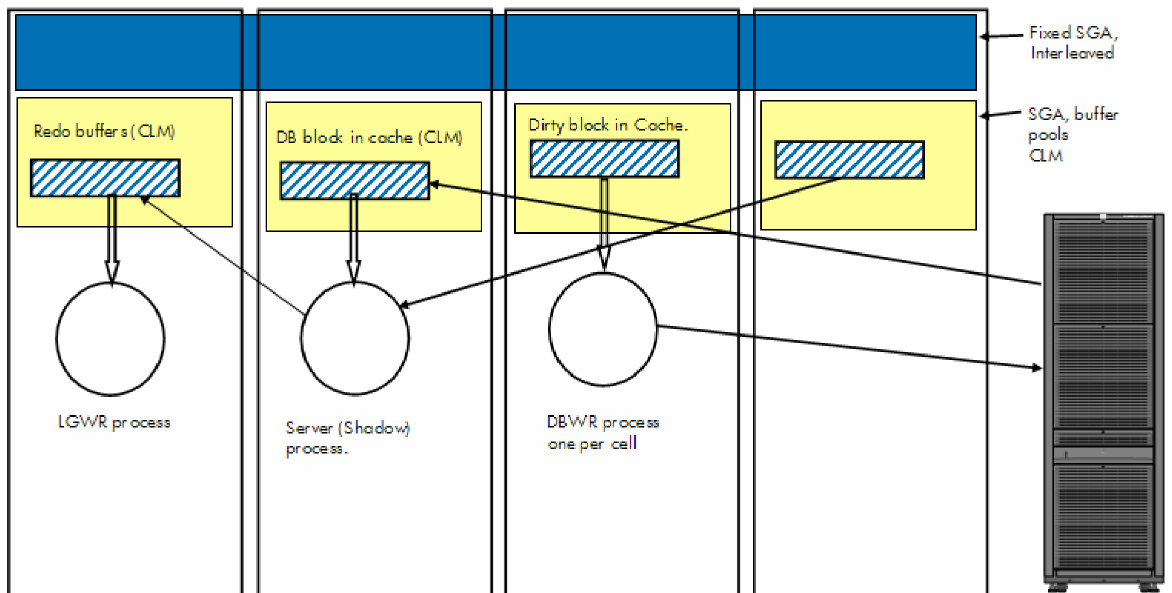
the operating system for optimal placement. On HP-UX, the default process placement on ccNUMA systems allows processes to migrate between cells. The following sections describe various methods to give processes launch policies that keep them locked in the locality domain they are started in. This will optimize memory accesses and improve performance. Note that the applicability of these methods are highly application dependent, and may not work in all cases. In most cases running Oracle in its default mode will give more than enough performance, and provide for a more seamless integration with HP-UX workload management tools.

The following document describes the methods of ccNUMA optimization on HP-UX in more detail: http://h20219.www2.hp.com/hpux11i/downloads/NUMAtuningGettingt.pdf.

## SGA and background processes

The Oracle ccNUMA architecture is outlined in Figure 7.

**Figure 7**. Oracle ccNUMA architecture




The architecture includes the following components:

- **SGA**
  The fixed System Global Area (SGA) is laid out in interleaved memory across all locality domains running the Oracle instance. Since this area contains data structures shared by all Oracle processes, it makes sense to store them in interleaved memory that does not favor any particular process.

  The remainder of the SGA (buffer pools, Java™ pool, shared pool, and so on) is evenly split up between locality domains in Cell Local Memory (CLM).
- **DBWRs**
  There is one database writer (DBWR) process per locality domain, each being responsible for flushing and coalescing blocks within its own locality domain.
- **LGWR**
  The log writer (LGWR) process runs in a single locality domain and is never allowed to migrate outside that domain.

- **Redo log buffers**
  Redo log buffers are allocated within the local portion of SGA, providing the fastest possible access to these critical resources.

The separate NUMA pool architecture has some of the same characteristics as running Oracle in a clustered environment - Real Application Clusters (RAC). In a RAC environment data (blocks) are shipped to the node where there is processing of that data, in ccNUMA it is assumed that the overhead of remote block accesses is small enough, so that blocks can be accessed remotely, and not duplicated in the local NUMA pools. Since blocks are not duplicated, there will be a performance impact, when accessing remote blocks, since memory latencies are higher when accessing remote locality domains.

It is up to the administrator to make sure that the system is configured with enough CLM to hold the SGA. A rough formula for CLM size is:

**CLM = SGA size / number of locality domains.**

This is the absolute minimum, in most cases enough CLM should be configured to hold the private memory of all Oracle processes as well. This is especially important, when process placement is controlled with launch policies locking processes to locality domains, not allowing process migration between locality domains. The amount of CLM is always applications dependent, but a reasonable guideline is to configure 7/8 of all memory as CLM.

Oracle's ccNUMA features are enabled by default in a cell-based system. ccNUMA can be disabled by setting the following static parameters:

```
_ENABLE_NUMA_OPTIMIZATIONS = FALSE
_DB_BLOCK_NUMA = 1
```

### Background processes

The LGWR and DBWR processes shown in Figure 7 are Oracle background processes, not actual server processes. Oracle controls placement of its background processes by locking them into appropriate locality domains. Oracle also spawns one database writer process per locality domain, each DBWR process being responsible for flushing out dirty blocks from its local NUMA pool only.

Other processes will depend on the ccNUMA-awareness of the OS to do process placement. The default launch policy in HP-UX is 'none', allowing processes to migrate between locality domains in high load situations. The following sections outline Oracle's scheduling of the other processes involved in running an Oracle instance (private server processes, Oracle shared servers, and parallel query slaves).

## Shadow processes.

Private server processes (shadow processes) are created and terminated as users connect to and disconnect from the database.

As shown in Figure 7, the data blocks required by a shadow process are allocated within the locality domain in which the process was initially started. Under the default launch policy, the shadow process may migrate to another domain. This has the advantage of keeping the load evenly distributed across the domains, but the disadvantage of block accesses to become remote, until the block is flushed out from the cache.

It is possible to evenly distribute shadow processes across locality domains, and locking them into domains by giving them the appropriate launch policies, as shown in an example in the later Tuning a ccNUMA architecture by localizing memory accesses section. However, preventing shadow processes from migrating to another locality domain may not always be a good choice in that some cells may become underutilized if some users exit quickly while others are running longer jobs. Oracle relies on the ccNUMA aware HP-UX scheduler to manage shadow processes, and does not do

explicit placement for those processes. If a process does migrate to a remote locality domain, it will loose its close proximity to its memory, both the database blocks in the local NUMA pool, and its private memory if it is running in CLM. As the process accesses database blocks not yet in memory, those shared memory accesses will become local, but the private data will not become local, until the process migrates back to its original domain.

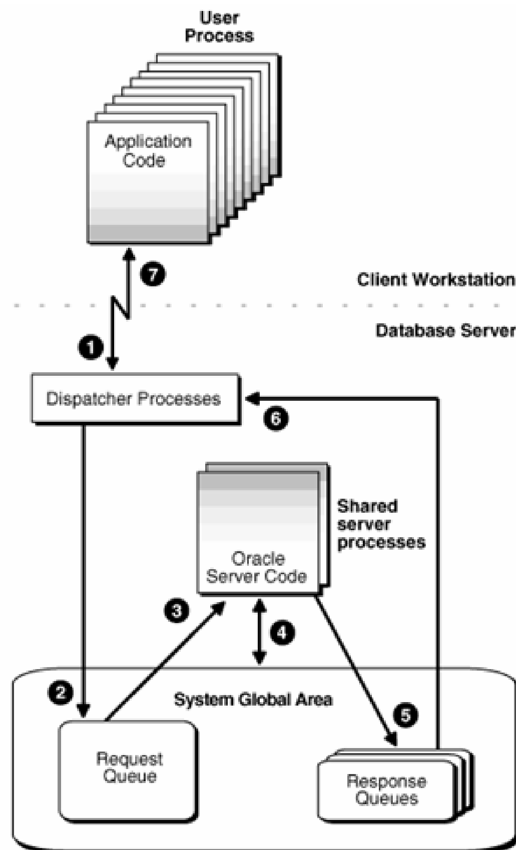
## Oracle shared servers

Oracle allows database users to share a single server through a mechanism known as a shared server (or multi-threaded server), which can be very useful in an environment where users log on, execute a short query, and log off. Since they are pre-spawned, using shared servers can eliminate the overhead needed to create new processes to serve connections.

Oracle shared servers are started by the database at database startup, not by the listener process, when connections to the database are created.

Figure 8 shows the shared server architecture.

**Figure 8**. Shared server architecture




Since the number of shared servers is fixed and all are started at database startup, they are good candidates for a ccNUMA environment. Oracle would only be required to spread the shared servers evenly across all locality domains and make sure they do not migrate between these domains.

Since a shared server stores part of its PGA (private memory) in the SGA, access to this memory would also be local. Sessions should also be restricted to shared servers within the same locality

domain. Currently, no Oracle release implements ccNUMA optimizations for shared servers, but future releases may change this.

## Parallel query slaves

Parallel query slaves are processes spawned to parallelize full table scans. The scan is split into pieces and each piece given to a query slave; these slaves work in parallel, resulting in much shorter query times.

Since parallel queries store their block buffers in process private memory, making sure that these areas are in CLM is a good practice. To make sure that slaves use CLM, make sure that you have enough CLM to hold the SGA and all process private memory of all Oracle processes. There is no simple rule for this, as the amount of CLM will be highly application dependent.

Parallel Query Slaves communicate through shared memory, and in the case of a complex query there may be a significant amount of messages passed between query slaves. Depending on the degree of parallelism, it could make sense to have a query run all its slaves in the same locality domain to make this shared memory access local. For queries with a very large degree of parallelism and a huge amount of query slaves running, it may makes sense to spread out the slaves across all locality domains. In any case, the user really has little control over query slave placement. The launch policy of query slaves may be inherited from the shadow process starting the slave. If the shadow process has the 'PACKED" policy, any slaves started by that shadow will inherit this policy and run in the same locality domain as the shadow. If the shadow has been given the round robin (RR) policy, slaves started by the shadow will round robin through the locality domains. If no policy has been specified (the default mode of operations) or if slaves are started at database startup, then the 'none' default launch policy is in effect for query slaves.

## ccNUMA and dynamic containers

As described earlier, Oracle ccNUMA optimizations strive to optimize performance by localizing memory accesses as much as possible; processes will run close to its private and shared data whenever possible. The Oracle layout assumes equally sized cells, Oracle shared memory is distributed evenly across all cells in use by Oracle. The number of processors in each cell should thus be the same as well. Of the different container types, nPars is a good fit for this type of architecture. This is also supported by the fact, that a ccNUMA instance is typically a large instance with a large number of processors. It is good practice to use as few cells as possible for the nPar that Oracle will run in, and make sure that these cells are as close as possible in the complex. Memory requirements may dictate how many cells need to be used. It is better to run Oracle in 2 cells with 8 processors than in 4 cells with 2 CPUs each.

It is not possible to run two instances of Oracle in a 4 cell nPar split up in two 2-cell CLM vPars, and move processors between those vPars to meet processor demands in the vPars. Oracle relies on processors running in the same locality domains that have NUMA pools allocated. This is not an absolute requirement, but the performance impact of adding a processor from a locality domain not belonging to the Oracle instance is difficult to predict.

Current Oracle implementations (10g and 11g) with NUMA optimizations enabled DO NOT support floating cells, so cells cannot be removed or added on the fly. Some of the reasons for this are:

- Oracle cannot extend its SGA into the memory in the floating cell. Oracle will not add a local piece of SGA in the new locality domain, nor will it start the cell local background processes in the new locality domain. This would also require a remapping of the SGA for all the processes that are running in the nPar.

- Current Oracle implementations are dependent on interleaved memory for the fixed SGA, so the cells Oracle starts up in cannot be made floating cells.
- Oracle may be able to take advantage of processors in a floating cell, although moving cells in and out of Oracle instances running in ccNUMA mode is not recommended.

Temporary Instant Capacity (TiCAP) CPUs (installed in the cells occupied by the Oracle instance) can potentially be used to add processing capacity to an Oracle instance running in ccNUMA mode. However, the Oracle architecture is really dependent on equally sized locality domains, adding a CPU to one locality domain may result in an unbalanced system, and is thus not recommended.

If Oracle runs without ccNUMA optimizations with memory fully interleaved, the processors in a floating cell will contribute to the Oracle workload, however all memory accesses to the Oracle shared memory areas (SGA) will be remote. The memory in a floating cell can not be allocated to the Oracle SGA.

The typical ccNUMA oracle deployment will have Oracle running in a static nPar, optimally laid out across as few locality domains as possible to meet Oracle's memory requirements.

## HP-UX 11.31 special considerations

On HP-UX 11i v3, a kernel parameter, 'numa_policy', can be used to control the allocation of memory. In environments, where Oracle is running with NUMA optimizations enabled, this parameter should be set to '1'. When Oracle is running in interleaved mode, this parameter should be set to '0'; otherwise the Oracle SGA will be laid out in CLM with unknown performance implications as a result.

# Tuning a ccNUMA architecture by localizing memory accesses

Since tuning opportunities with Oracle on ccNUMA tend to focus on maximizing local memory accesses, this section describes how to configure an Oracle instance to improve performance by maximizing the amount of CLM accesses.

As noted earlier, no current version of Oracle does placement of server (shadow processes); no HP-UX launch policies are set by Oracle, nor are those processes explicitly locked into locality domains.

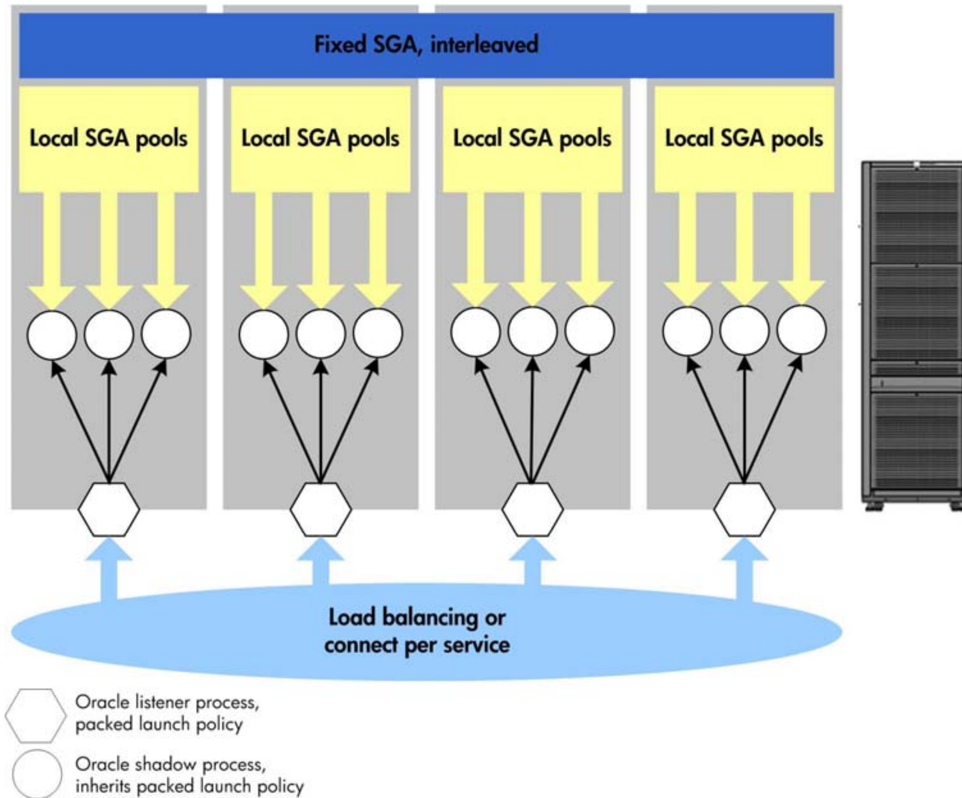
As the method described here locks all processes into locality domains, it does not lend itself to dynamic reconfiguration of the container Oracle runs in. It is important to have equally sized cells in the nPar Oracle is running in.

Memory access times constitute the main component of the cycles per instruction (CPI) required by Oracle to execute a typical workload. As a result, anything that can be done to minimize this component improves performance.

The drawback of the methods described in this section is that Oracle processes will be locked into the locality domains they are started in. This is good from a memory access point of view, since all memory accesses to process private data will be local, but the processes are not allowed to migrate out from their locality domains. This could obviously result in an unbalanced system as users log out and log in. Therefore, these methods work best where there is a pre-defined amount of processes running. Pooled servers and shared servers are good candidates for this kind of optimizations.

Figure 9 shows an example of achieving maximum memory locality in a ccNUMA-based server.

**Figure 9**. Oracle ccNUMA tuning




To make sure that Oracle shadow processes never migrate between locality domains, you must set appropriate HP-UX launch policies. Normally, this could be a difficult task as potentially thousands of processes will be executing and terminating under Oracle control; moreover, Oracle does not set launch policies. However, since launch polices are inherited, you simply need to set the appropriate policy for the Oracle listener process, which is responsible for spawning the shadow processes on incoming connections.

This configuration assumes the use of private Oracle server processes, which is the most common way of connecting to a database, especially in benchmarking environments. There is no way of efficiently controlling the placement of shared servers, as these will be started by the database at database startup.

The following steps provide an example of configuring an Oracle instance to improve performance by maximizing the amount of CLM accesses:

1. Configure the Oracle network configuration file, 'listener.ora,' for multiple listeners. In a four-cell system you would configure four listeners (listener1 – listener4, for instance). A sample 'listener.ora' file follows:

**Note:**
Consult the appropriate Oracle documentation for information on setting up the Oracle network configuration file.

```
LISTENER1 =
(DESCRIPTION_LIST =
   (DESCRIPTION =
      (ADDRESS_LIST =
          (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC0))
       )
      (ADDRESS_LIST =
          (ADDRESS = (PROTOCOL = TCP)(HOST = host1)(PORT = 1521))
       )
    )
 )

# SID list of the listener listener1
SID_LIST_LISTPROD1 =
   (SID_LIST =
      (SID_DESC =
         (SID_NAME = PLSExtProc)
         (ORACLE_HOME = /oracle/ora10g)
         (PROGRAM = extproc)
       )
      (SID_DESC =
         (GLOBAL_DBNAME = orcl10g)
         (ORACLE_HOME = /oracle/ora10g)
         (SID_NAME = orcl10g)
       )
    )

# listprod2 is the name of the second  listener
LISTENER2 =
(DESCRIPTION_LIST =
   (DESCRIPTION =
      (ADDRESS_LIST =
         (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC0))
       )
      (ADDRESS_LIST =
         (ADDRESS = (PROTOCOL = TCP)(HOST = host1)(PORT = 1526))
      )
   )
)

# SID list of the listener listprod1
SID_LIST_LISTPROD2 =
   (SID_LIST =
      (SID_DESC =
         (SID_NAME = PLSExtProc)
         (ORACLE_HOME = /oracle/ora10g)
         (PROGRAM = extproc)
       )

     (SID_DESC =
        (GLOBAL_DBNAME = orcl10g)
        (ORACLE_HOME = /oracle/ora10g)
        (SID_NAME = orcl10g)
      )

<< repeat for listener3 and listener4 >>
)
```

- Start the listener processes, giving them the packed launch policy. This example assumes four cells:

```
mpsched -l 1 -P PACKED lsnrctl start listener0
mpsched -l 2 -P PACKED lsnrctl start listener1
mpsched -l 3 -P PACKED lsnrctl tart listener3
mpsched -l 4 -P PACKED lsnrctl start listener4
```

Use either Oracle load balancing or a transaction monitor to evenly distribute connections to the listener processes. The listeners spawn off shadow processes that inherit the packed policy and, thus, can only run within the locality domain in which they started, with optimal access to memory.

This methodology can be used to distribute different services within the same database instance, providing a useful mechanism for setting up services. If listeners were set up to handle different services, then each service would be restricted to its own locality domain(s), achieving good separation between the services. In addition, since Oracle tries to keep the database block buffers close to the shadow processes, there would be minimal buffer cache contention between services.

In general, this kind of setup is very good for partitioned applications where the working set of each connection fits in the local buffer cache. However, there can be serious side-effects, if the processes spawn parallel query slaves, as these will all inherit the PACKED launch policy, and run in the same locality domain. This may or may not be a good thing, depending on the degree of parallelism of the query.

Note, that not all applications lend themselves to a multi-listener configuration like this, and can be run only in a single listener configuration.

## Tuning with a single listener

In some cases, it is not practical to use many listeners. The listener process is still the only way of controlling placement of server processes. In the case of a single listener process (or less listener processes than locality domains in the system), you can still control placement by giving the listener either the Round Robin launch policy (RR) or  the Least Loaded launch policy (LL):

```
mpsched -l 3 -P RR lsnrctl start
mpsched -l 4 -P LL lsnrctl start
```

The Round Robin policy will round robin the processes to be started through the locality domains, the LL policy will look for the "least loaded" locality domain to start the server process in. Processes started with either of these launch policies will never migrate out of the domain they are started in, thus providing maximum locality for their memory accesses.

If the parallel query slaves are started by a shadow process, they will also using the round robin policy; this may not always be optimal from a performance point of view.

This method also works best, when Oracle runs in a static container, as the load is evenly distributed across locality domains with the round robin (RR) launch policy.

## Tuning with psets

It may be tempting to use processor sets (psets) to control placement of Oracle processes, splitting up the Oracle instance in several psets. For instance, there could be a pset per locality domain, or even smaller psets making sure all processors in the pset share the same memory busses for optimal performance. However this method is not recommended, for the following reasons:

- Oracle would not know the correct amount of processors it has access to. The user would have to force the correct setting for cpu_count by setting this parameter every time the number of CPUs changes in the psets running the Oracle instance.
- Oracle may have the incorrect view of the amount of locality domains it has access to.
- Oracle background processes may be pushed out of the locality domains Oracle wants to run them in.

## Oracle and ccNUMA, Summary

ccNUMA is an evolving architecture both at the system level, OS level and at the Oracle level. The architecture leaves the Oracle user with many alternatives for deploying Oracle on a large server. Running oracle with ccNUMA optimizations enabled, will attempt to use the ccNUMA architecture to its advantage by localizing memory accesses as much as possible. A static nPar is the best choice for running Oracle with ccNUMA optimizations enabled.

Performance can be further improved by setting launch policies for server processes, guaranteeing that a server process never migrates from the locality domain it was started in. The 'PACKED' or 'RR' launch policies can be used to control the placement of server processes. Launch policies are inherited by process children, for instance query slaves spawned by a query coordinator will inherit its launch policy. If the coordinator has the PACKED policy, all query slaves will run in the same locality domain as the coordinator, in the case of the 'RR' policy, the query slaves will be distributed evenly across all locality domains.

HP further recommends that the nPar is configured according to the following rules:

- Minimize the number of locality domains needed.
- Configure a symmetric amount of memory in each locality.
- Configure a symmetric amount of processors in each locality domain.
- Configure 7/8 CLM. Note, that if Oracle is running with NUMA optimizations enabled, enough CLM has to be configured to hold Oracle's NUMA pools.

Oracle on ccNUMA works best with a partitionable workload, memory accesses should be local to the highest possible degree. Some applications do not fit into this class, and the performance benefit of ccNUMA diminishes, as many Oracle database block accesses become remote. In this kind of environment, Oracle can be run with NUMA optimizations disabled, using interleaved memory. This would also make it possible to utilize all the automatic reconfiguration features currently supported by tools like gWLM and WLM, and that have been certified in Oracle environments (these tools are not currently ccNUMA aware).

In most cases, turning off Oracle's ccNUMA optimizations by setting the two parameters discussed earlier, will provide enough performance and is the easiest way of adapting to the dynamic features of HP-UX. In this case, the server should be configured with a minimum amount of cell local memory, 10% is likely a good rule of thumb to let the operating system take advantage of its ccNUMA optimizations, still allowing Oracle to run in interleaved mode.

# Conclusion

The 10*g* and 11*g* versions of the Oracle database have included features to optimize performance in systems with non-uniform memory access, such as the HP Integrity servers. Utilizing this architecture can provide performance gains for performance critical applications. The architecture can be further boosted by letting HP-UX control launch policies for Oracle shadow processes. In many cases this kind of optimization is not needed, and applications are well served by Oracle's default mode of operation. Oracle can also be executed in interleaved mode, simulating a large SMP machine; this will provide lots of flexibility in dynamically reconfiguring partitions running Oracle.

The Oracle DBRM (Database resource manager) was also described, and it was shown, that some integration with the HP workload management product is possible, letting the Oracle DBRM take care of the Oracle internal resources.

# For more information

HP-UX, www.hp.com/go/hpux

HP ActiveAnswers, www.hp.com/solutions/activeanswers

HP Integrity servers, www.hp.com/go/integrity

Oracle Database http://www.oracle.com/database/index.html

To help us improve our documents, please provide feedback at www.hp.com/solutions/feedback

© 2008 Hewlett-Packard Development Company, L.P. The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Java is a US trademark of Sun Microsystems, Inc. UNIX is a registered trademark of The Open Group.

4AA2-0547ENW, June 2008